程序 14-28 linux/include/linux/tty.h

```
/*
 * 'tty.h' defines some structures used by tty_io.c and some defines.
 *
 * NOTE! Don't touch this without checking that nothing in rs_io.s or
 * con_io.s breaks. Some constants are hardwired into the system (mainly
 * offsets into 'tty_queue'
 */
/*
 * 'tty.h'中定义了 tty_io.c 程序使用的某些结构和其他一些定义。
 *
 * 注意！在修改这里的定义时，一定要检查 rs_io.s 或 con_io.s 程序中不会出现问题。
 * 在系统中有些常量是直接写在程序中的（主要是一些 tty_queue 中的偏移值）。
 */

#ifndef _TTY_H
#define _TTY_H

#define MAX_CONSOLES    8           // 最大虚拟控制台数量。
#define NR_SERIALS      2           // 串行终端数量。
#define NR_PTYS         4           // 伪终端数量。

extern int NR_CONSOLES;             // 虚拟控制台数量。

#include <termios.h>        // 终端输入输出函数头文件。主要定义控制异步通信口的终端接口。

#define TTY_BUF_SIZE 1024       // tty 缓冲区（缓冲队列）大小。

// tty 字符缓冲队列数据结构。用于 tty_struc 结构中的读、写和辅助（规范）缓冲队列。
struct tty_queue {
        unsigned long data;             // 队列缓冲区中含有字符行数值（不是当前字符数）。
                                        // 对于串口终端，则存放串行端口地址。
        unsigned long head;             // 缓冲区中数据头指针。
        unsigned long tail;             // 缓冲区中数据尾指针。
        struct task_struct * proc_list; // 等待本队列的进程列表。
        char buf[TTY_BUF_SIZE];         // 队列的缓冲区。
};

#define IS_A_CONSOLE(min)       (((min) & 0xC0) == 0x00)        // 是一个控制终端。
#define IS_A_SERIAL(min)        (((min) & 0xC0) == 0x40)        // 是一个串行终端。
#define IS_A_PTY(min)           ((min) & 0x80)                  // 是一个伪终端。
#define IS_A_PTY_MASTER(min)    (((min) & 0xC0) == 0x80)        // 是一个主伪终端。
#define IS_A_PTY_SLAVE(min)     (((min) & 0xC0) == 0xC0)        // 是一个辅伪终端。
#define PTY_OTHER(min)          ((min) ^ 0x40)                  // 其他伪终端。

// 以下定义了 tty 等待队列中缓冲区操作宏函数。（tail 在前，head 在后，参见 tty_io.c 的图）。
// a 缓冲区指针前移 1 字节，若已超出缓冲区右侧，则指针循环。
#define INC(a) ((a) = ((a)+1) & (TTY_BUF_SIZE-1))
// a 缓冲区指针后退 1 字节，并循环。
#define DEC(a) ((a) = ((a)-1) & (TTY_BUF_SIZE-1))
// 清空指定队列的缓冲区。
#define EMPTY(a) ((a)->head == (a)->tail)
// 缓冲区还可存放字符的长度（空闲区长度）。
```

```
40 #define LEFT(a) (((a)->tail-(a)->head-1)&(TTY_BUF_SIZE-1))
   // 缓冲区中最后一个位置。
41 #define LAST(a) ((a)->buf[(TTY_BUF_SIZE-1)&((a)->head-1)])
   // 缓冲区满（如果为 1 的话）。
42 #define FULL(a) (!LEFT(a))
   // 缓冲区中已存放字符的长度（字符数）。
43 #define CHARS(a) (((a)->head-(a)->tail)&(TTY_BUF_SIZE-1))
   // 从 queue 队列项缓冲区中取一字符(从 tail 处，并且 tail+=1)。
44 #define GETCH(queue,c) \
45 (void)({c=(queue)->buf[(queue)->tail];INC((queue)->tail);})
   // 往 queue 队列项缓冲区中放置一字符（在 head 处，并且 head+=1）。
46 #define PUTCH(c,queue) \
47 (void)({(queue)->buf[(queue)->head]=(c);INC((queue)->head);})
48
   // 判断终端键盘字符类型。
49 #define INTR_CHAR(tty) ((tty)->termios.c_cc[VINTR])     // 中断符。发中断信号 SIGINT。
50 #define QUIT_CHAR(tty) ((tty)->termios.c_cc[VQUIT])     // 退出符。发退出信号 SIGQUIT。
51 #define ERASE_CHAR(tty) ((tty)->termios.c_cc[VERASE])   // 削除符。擦除一个字符。
52 #define KILL_CHAR(tty) ((tty)->termios.c_cc[VKILL])     // 删除行。删除一行字符。
53 #define EOF_CHAR(tty) ((tty)->termios.c_cc[VEOF])       // 文件结束符。
54 #define START_CHAR(tty) ((tty)->termios.c_cc[VSTART])   // 开始符。恢复输出。
55 #define STOP_CHAR(tty) ((tty)->termios.c_cc[VSTOP])     // 停止符。停止输出。
56 #define SUSPEND_CHAR(tty) ((tty)->termios.c_cc[VSUSP])  // 挂起符。发挂起信号 SIGTSTP。
57
   // tty 数据结构。
58 struct tty_struct {
59         struct termios termios;                   // 终端 io 属性和控制字符数据结构。
60         int pgrp;                                 // 所属进程组。
61         int session;                              // 会话号。
62         int stopped;                              // 停止标志。
63         void (*write)(struct tty_struct * tty);   // tty 写函数指针。
64         struct tty_queue *read_q;                 // tty 读队列。
65         struct tty_queue *write_q;                // tty 写队列。
66         struct tty_queue *secondary;              // tty 辅助队列(存放规范模式字符序列)，
67         };                                        // 可称为规范(熟)模式队列。
68
69 extern struct tty_struct tty_table[];             // tty 结构数组。
70 extern int fg_console;                            // 前台控制台号。
71
   // 根据终端类型在 tty_table[] 中取对应终端号 nr 的 tty 结构指针。第 73 行后半部分用于
   // 根据子设备号 dev 在 tty_table[]表中选择对应的 tty 结构。如果 dev = 0，表示正在使用
   // 前台终端，因此直接使用终端号 fg_console 作为 tty_table[] 项索引取 tty 结构。如果
   // dev 大于 0，那么就要分两种情况考虑：① dev 是虚拟终端号；② dev 是串行终端号或者
   // 伪终端号。对于虚拟终端其 tty 结构在 tty_table[]中索引项是 dev-1（0 -- 63）。对于
   // 其它类型终端，则它们的 tty 结构索引项就是 dev。例如，如果 dev = 64，表示是一个串
   // 行终端 1，则其 tty 结构就是 ttb_table[dev]。 如果 dev = 1，则对应终端的 tty 结构是
   // tty_table[0]。参见 tty_io.c 程序第 70 -- 73 行。
72 #define TTY_TABLE(nr) \
73 (tty_table + ((nr) ? (((nr) < 64)? (nr)-1:(nr)) : fg_console))
74
   // 这里给出了终端 termios 结构中可更改的特殊字符数组 c_cc[]的初始值。该 termios 结构
   // 定义在 include/termios.h 中。POSIX.1 定义了 11 个特殊字符，但是 Linux 系统还另外定
   // 义了 SVR4 使用的 6 个特殊字符。如果定义了_POSIX_VDISABLE（\0），那么当某一项值等
```

```
    // 于_POSIX_VDISABLE 的值时，表示禁止使用相应的特殊字符。[8 进制值]
 75 /*      intr=^C         quit=^|         erase=del       kill=^U
 76         eof=^D          vtime=\0        vmin=\1         sxtc=\0
 77         start=^Q        stop=^S         susp=^Z         eol=\0
 78         reprint=^R      discard=^U      werase=^W       lnext=^V
 79         eol2=\0
 80 */
    /* 中断 intr=^C      退出 quit=^|      删除 erase=del      终止 kill=^U
     * 文件结束 eof=^D   vtime=\0          vmin=\1           sxtc=\0
     * 开始 start=^Q     停止 stop=^S      挂起 susp=^Z      行结束 eol=\0
     * 重显 reprint=^R   丢弃 discard=^U   werase=^W         lnext=^V
     * 行结束 eol2=\0
     */
 81 #define INIT_C_CC "\003\034\177\025\004\0\1\0\021\023\032\0\022\017\027\026\0"
 82
 83 void rs_init(void);        // 异步串行通信初始化。(kernel/chr_drv/serial.c)
 84 void con_init(void);       // 控制终端初始化。     (kernel/chr_drv/console.c)
 85 void tty_init(void);       // tty 初始化。         (kernel/chr_drv/tty_io.c)
 86
 87 int tty_read(unsigned c, char * buf, int n);   // (kernel/chr_drv/tty_io.c)
 88 int tty_write(unsigned c, char * buf, int n);  // (kernel/chr_drv/tty_io.c)
 89
 90 void con_write(struct tty_struct * tty);       // (kernel/chr_drv/console.c)
 91 void rs_write(struct tty_struct * tty);        // (kernel/chr_drv/serial.c)
 92 void mpty_write(struct tty_struct * tty);      // (kernel/chr_drv/pty.c)
 93 void spty_write(struct tty_struct * tty);      // (kernel/chr_drv/pty.c)
 94
 95 void copy_to_cooked(struct tty_struct * tty);  // (kernel/chr_drv/tty_io.c)
 96
 97 void update_screen(void);                      // (kernel/chr_drv/console.c)
 98
 99 #endif
100
```